InCore Development Manual

Release 2.3.0

in.hub GmbH

Feb 11, 2021

Contents

1 Introduction

This guide will help you getting started with the InCore Framework Software Development Kit (SDK) and developing, running and debugging your first application with the Qt Creator IDE. If your development environment is already set up, you can continue with the InCore tutorials.

The InCore Framework offers ready-to-use high-level software blocks ("objects") for building IoT/IIoT applications quickly. InCore Applications configure and link InCore objects based on a declarative JSON-based syntax using the Qt Modeling Language (QML). Custom operations and processes can be implemented in JavaScript functions easily.

Before you can start developing apps for the HUB-GM100 you need to ensure to have installed all necessary software components of the toolchain, and the connection between your PC and the gateway is configured properly. Therefore please study the InCore Install Guide and ensure to complete all the described steps.

Please also refer to the in.hub website and the HUB-GM100 product site for additional information and help.

2 Related documentation and resources

2.1 InCore

- Online documentation: https://incore.readthedocs.io/
- InCore Install Guide: https://incore.readthedocs.io/en/latest/installation

2.2 HUB-GM100 resources

- Product site: https://inhub.de/produkt/iot-gateway-hub-gm100
- Data sheet: https://inhub.de/files/product-docs/gs-datasheet-hub-gm100-11-2018.pdf

2.3 Qt related resources

- Getting Started with Qt: https://doc.qt.io/qt-5/gettingstarted.html
- Qt Creator online documentation: https://doc.qt.io/qtcreator/index.html
- Qt QML Getting Started: https://doc.qt.io/qt-5/qtqml-index.html

3 Creating, deploying and running applications

3.1 Creating an example application

After performing the necessary installation and configuration steps as described in the InCore Install Guide, you can now start the application development.

First start Qt Creator, open the **File** menu and select **New File or Project**. In the appearing dialog box select **Other Project** from the left handed menu and the **Empty qmake Project** template (Fig. 3.1).

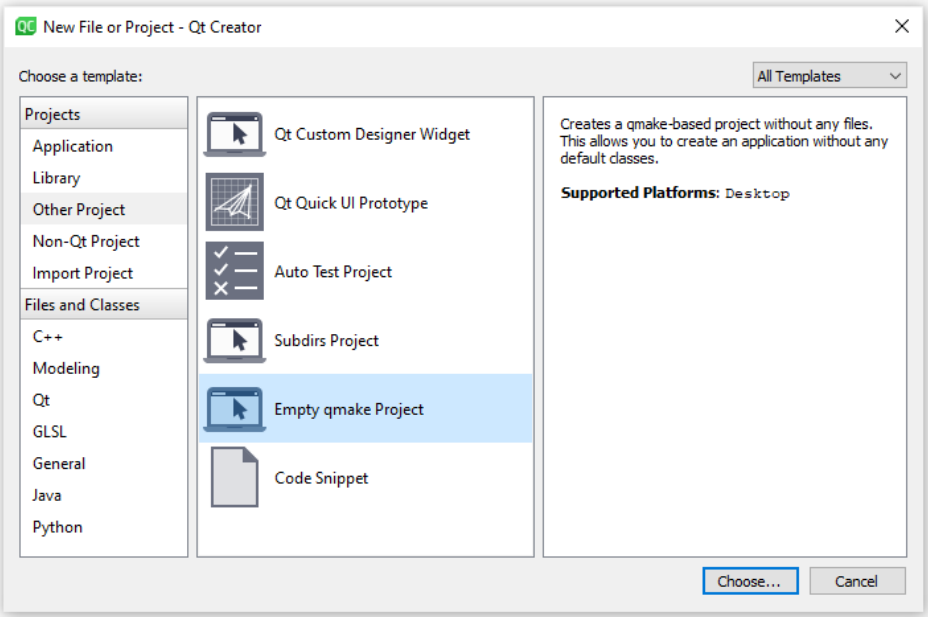

Figure3.1: Creating a new project in Qt Creator

In the appearing dialog box (Fig. 3.2), give your project the name `test` and choose its location.

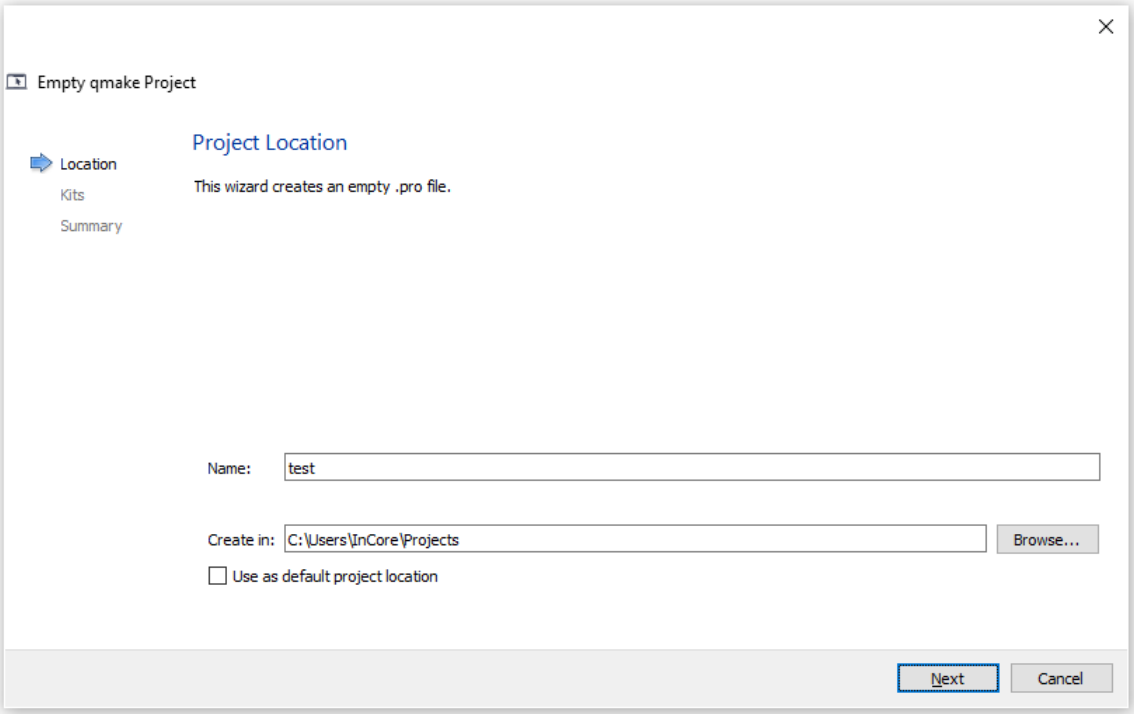

Figure3.2: Setting project name and location in Qt Creator

On the next page please select the kit you've created in one of the steps described in the InCore Install Guide (Fig. 3.3). Please press **Finish** on the next page.

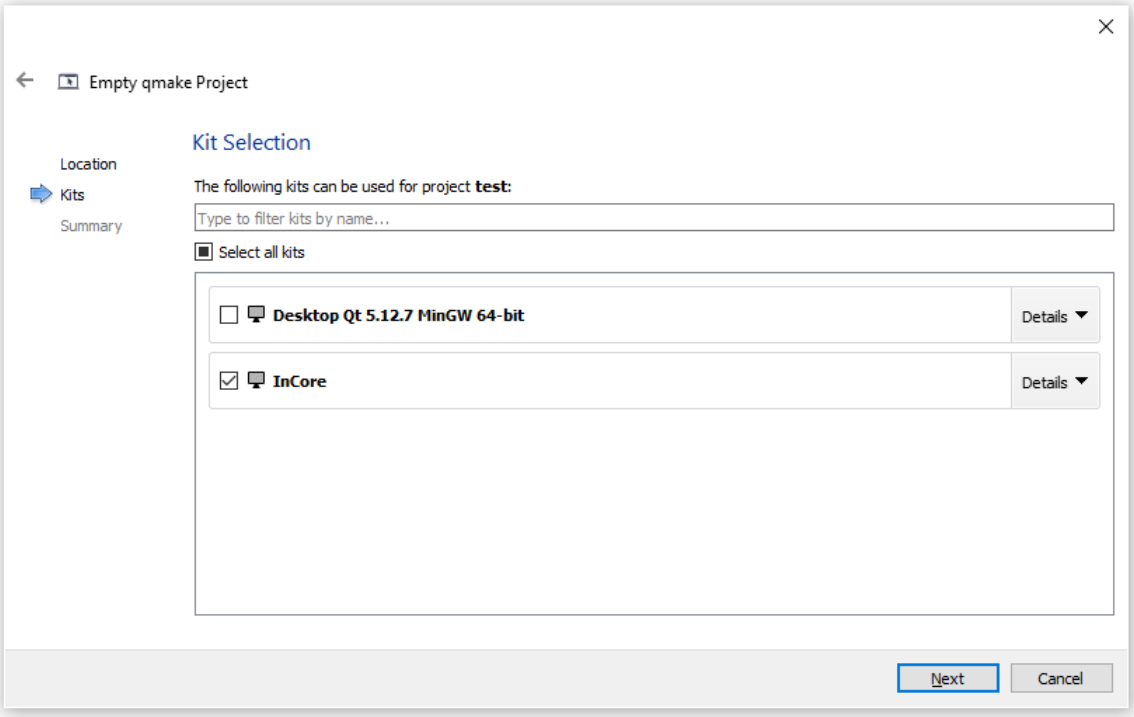

Figure3.3: Selecting kit for new project in Qt Creator

Now the project is created and Qt Creator should automatically open the project file `test.pro`. Replace all existing lines with the following and save it:

```
CONFIG = incore_app
TARGET = test
```

Now please add a new file to the project via the menu **File** → **New File or Project** and select **Qt** → **QML File** in the appearing dialog box (Fig. 3.4). Give it the name `main.qml`, press **Next** and **Finish** afterwards.

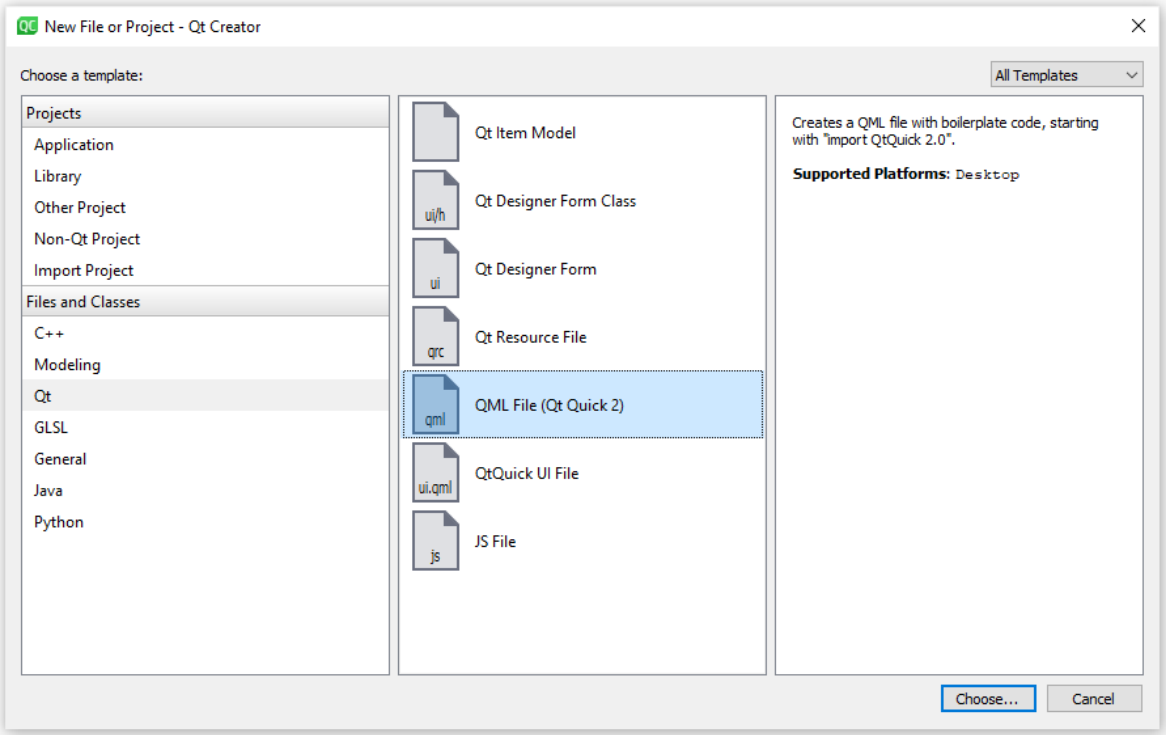

Figure3.4: Creating a new QML file in Qt Creator

Qt Creator automatically opens the new file `main.qml` in the text editor. Please remove the example code lines since they are not required for InCore apps.

Every InCore app has to import the InCore modules it wants to use. The basic module which is always required is called `Foundation`. To import it add the following line to `main.qml`:

```
import InCore.Foundation 2.0
```

An application should use Application as its root object:

```
Application {
}
```

Of course we want to start with a classic, so the final `main.qml` should look like this:

```
import InCore.Foundation 2.0

Application {
    onCompleted: console.log("Hello world!")
}
```

In the next section this example application will be deployed and run on a HUB-GM100.

3.2 Deploying and running an application

In order to deploy the app to a HUB-GM100, the **Run Settings** of your project have to be configured accordingly. Choose **Projects** from the left handed menu and edit the **Run Settings** as shown in Fig. 3.5:

1) If not already selected, change the deployment method to **Remote Linux Host** or add a new deployment method,
2) Change the **Run configuration** to **Custom Executable (on HUB-GM100)**,
3) Change the **Remote executable** to `incore-cli`,
4) Change the **Command line arguments** to `run test`.

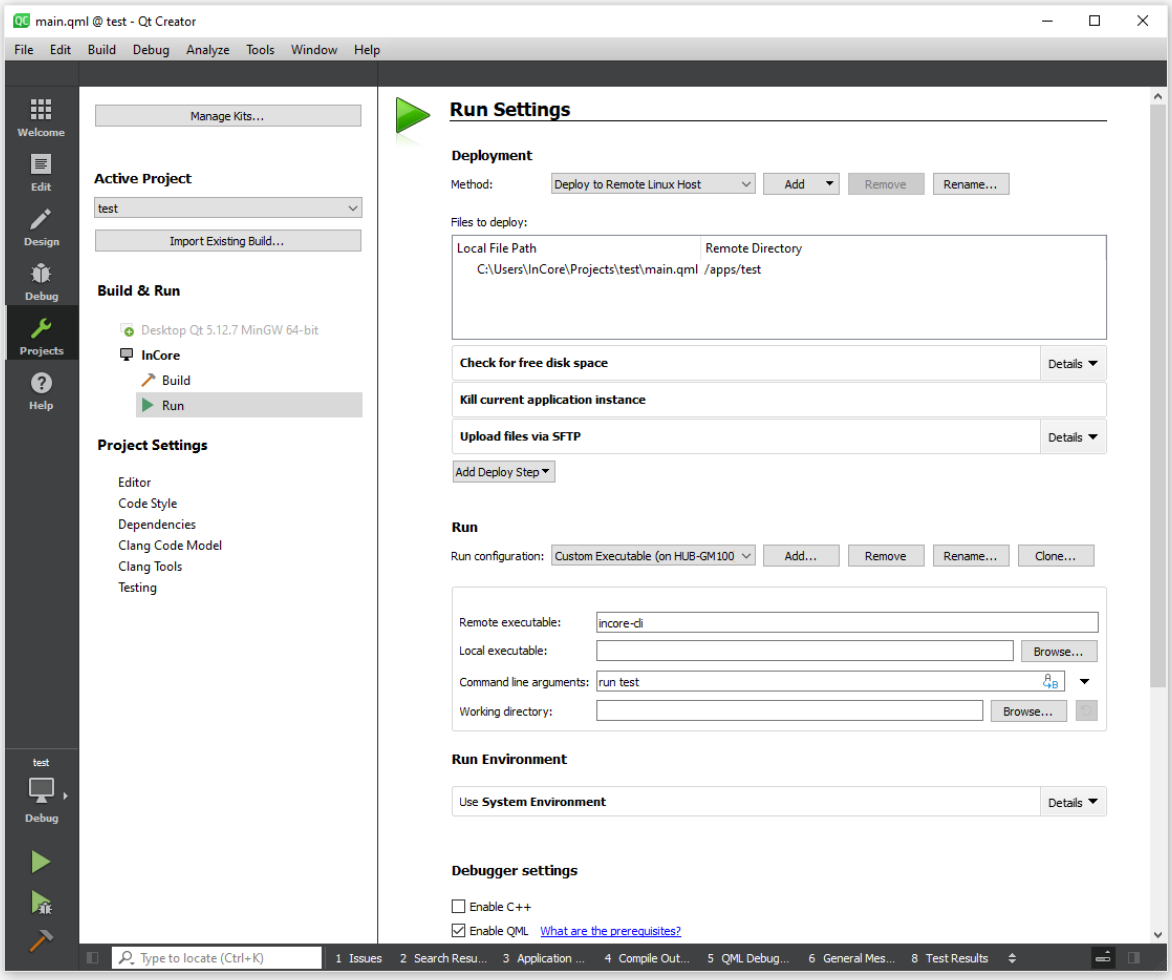

Figure3.5: Settings for running InCore apps on a HUB-GM100

Before the application can be deployed and started, the device needs to be put into development mode. In this mode the storage for apps is made writable, all running apps are stopped and the SSH service is being started. To activate development mode, start a PuTTY session and log in as described in section Logging in in the SIINEOS Manual. After logging in enter the command `incore-cli dev start` (Fig. 3.6).

The development mode can later be stopped again via `incore-cli dev stop` or by rebooting the device.

Important: The development mode is only activated temporarily. Once the device is rebooted or powered off, the device starts in regular non-development mode again.

Figure3.6: Starting development mode on a HUB-GM100

Now switch back to the editor window of the Qt Creator. The application can now be run by clicking **Build** → **Run** from the main menu or using the shortcut `Ctrl+R` or by pressing the green play button in the left handed menu.

If all steps have been followed properly, the application will be started and print its output to the IDE as shown in Fig. 3.7.

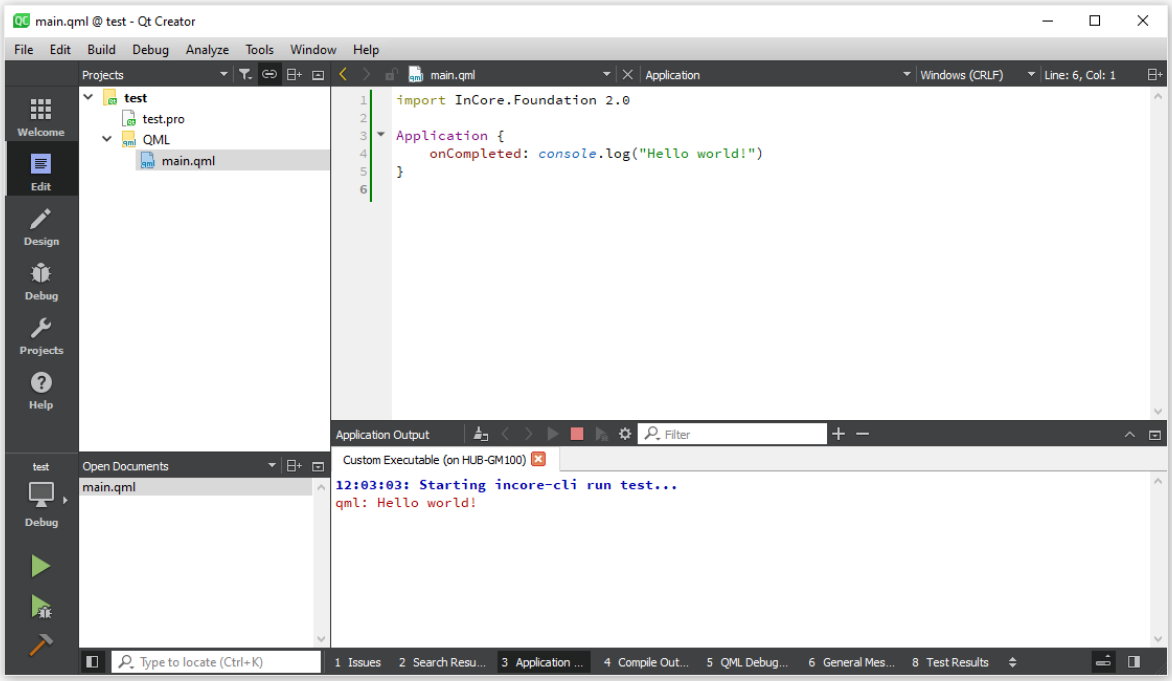

Figure3.7: Output of the test application running on a HUB-GM100

Note: Whenever an application is deployed, it is stored permanently in the `/apps/<APPNAME>` directory. It will be started automatically on every start of the HUB-GM100. In general when SIINEOS boots, it creates an application instance for every subdirectory inside `/apps` containing a file called `main.qml`.

Congratulations, you've successfully deployed and started your first application on a HUB-GM100! Now you can start playing with the HUB-GM100 and continue with the InCore tutorials.

4 Debugging applications

4.1 QML debugging

InCore apps can be debugged easily using the builtin QML debugger in Qt Creator. Before starting, the debugger settings have to be adjusted. Open the **Run Settings** of your project, scroll down to the **Debugger settings** and ensure **Enable C++** is not checked whereas **Enable QML** is (Fig. 4.1).

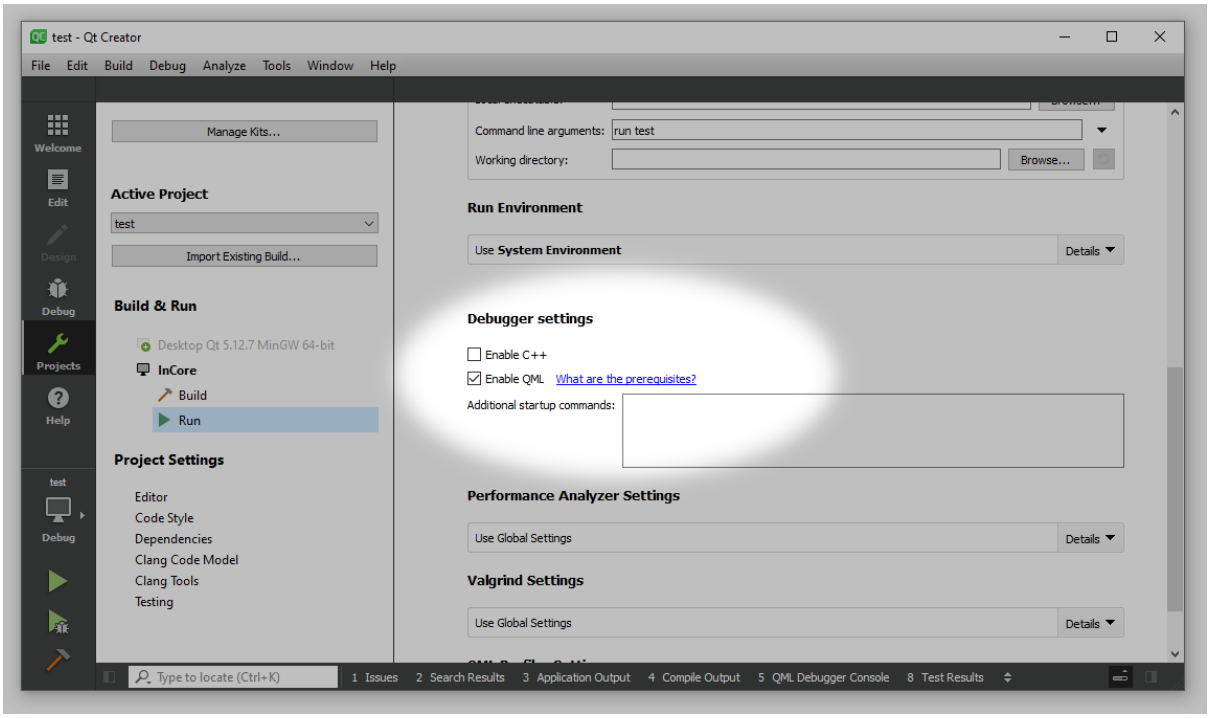

Figure4.1: Settings for debugging InCore apps on a HUB-GM100

Type the example code implementing a simple timer which increments a variable every second:

```
import InCore.Foundation 2.0

Application {
    Timer {
        property int foo: 0

        onTriggered: {
            foo++
        }
    }
}
```

Now start debugging by clicking the green debug button in the left handed menu or by pressing `F5`. The program will start running and you can inspect all properties in realtime through the **Locals** view on the right side. Expand `Application`, `Timer` and `Properties` to see `foo` being incremented every second (Fig. 4.2).

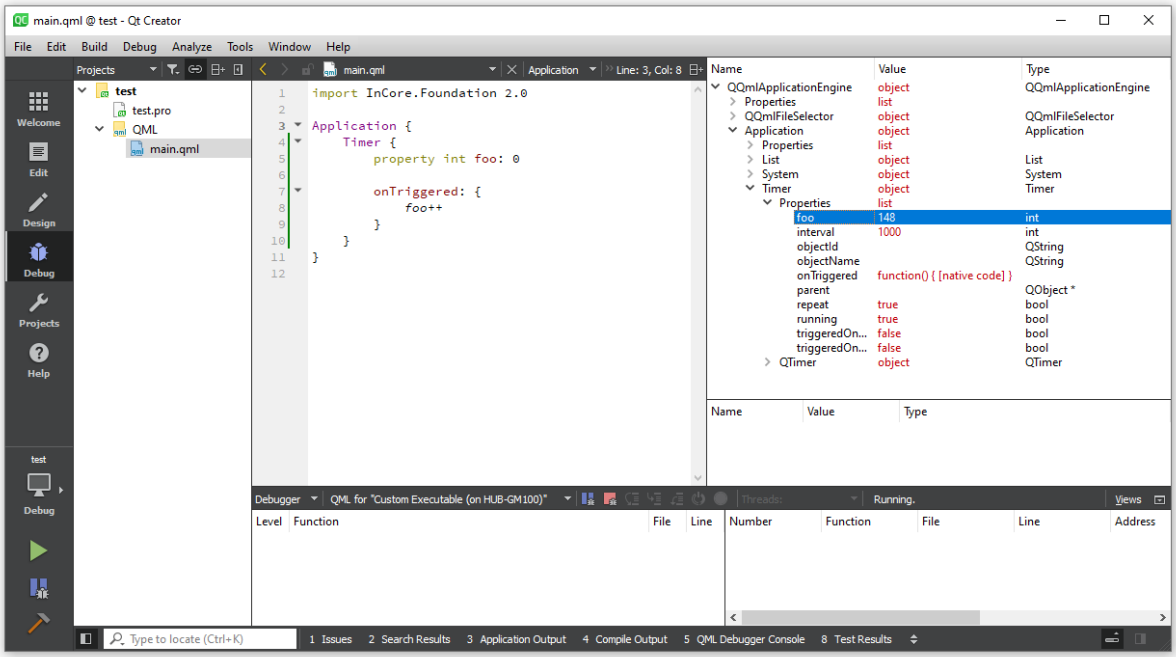

Figure4.2: Debugging an InCore app on a HUB-GM100

Next set a breakpoint at the code line that increments `foo` (click left to the line number). This will make the program being interrupted whenever this code line is executed (Fig. 4.3). You can inspect and even modify the properties of the local object (and navigate through the object hierarchy via the `parent` property). For example you can change the `interval` property and observe an appropriate behaviour change.

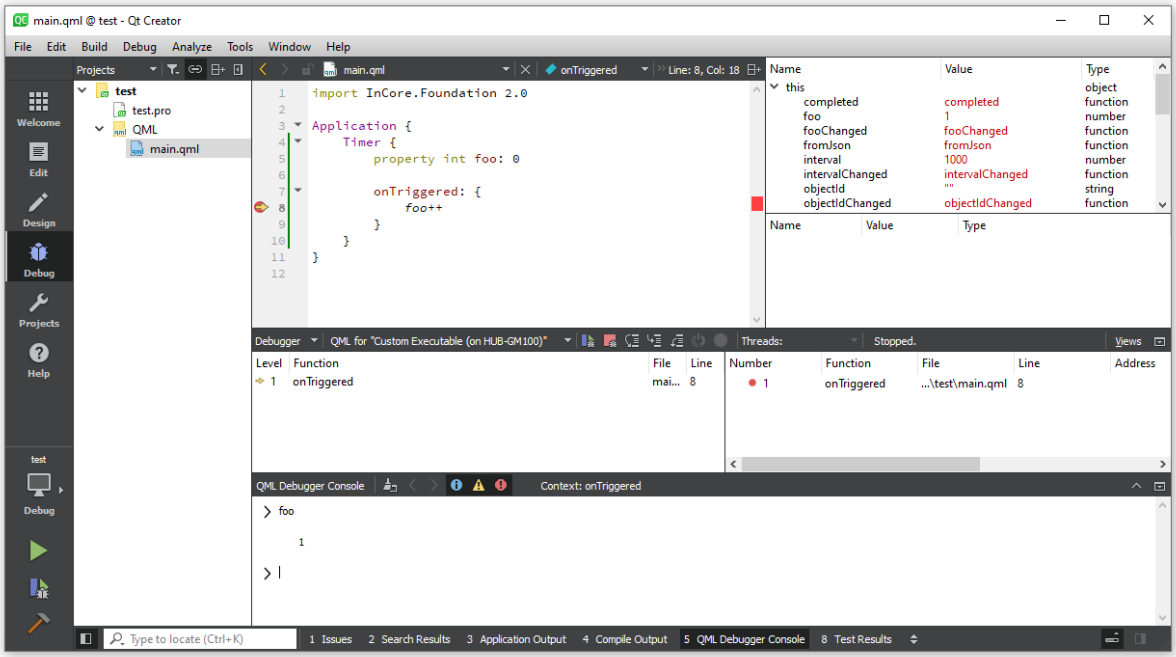

Figure4.3: Program execution interrupted at a breakpoint

Further information on debugging QML applications can be found in the Qt Creator Manual (ignore the parts specific to QML user interfaces).

4.2 InCore debugging

All InCore modules provide additional debug messages which can be enabled easily by setting the Application.debug property to `true`. When enabled, extra messages will be logged to the console, especially all error codes and strings whenever an error occurs in any object (which saves adding an `onErrorOccurred` handler to every object). Some additional messages also indicate code locations or invalid values causing the actual errors.

To debug applications without modifying them, you can alternatively set the `INCORE_DEBUG` environment variable prior to executing `incore-cli run ...`. Set the variable either at the command line when running `incore-cli` manually anyway or modify the project's **Run Environment** and add the corresponding environment variable.

To further increase the verbosity level, internal InCore function calls can be traced. This can be enabled by setting the Application.trace property to `true` or setting the `INCORE_TRACE` environment variable. When tracing is enabled, your console will likely be flooded with messages unless your application is very small or static. Consider setting a log message filter as described in section **Customizing message logging**.

4.3 Customizing message logging

In order to customize the format of log messages or add meta data fields such as a formatted time you can either modify the Application.messageLoggingPattern property or set the `QT_MESSAGE_PATTERN` environment variable accordingly (see section **InCore debugging** on how to do this in Qt Creator). See the Qt documentation on message patterns for details and all supported placeholders.

To filter messages you can set the Application.messageLoggingFilterRules property or set the `QT_LOGGING_RULES` environment variable. See the property documentation for details on syntax and supported values.

5 Accessing application data

Objects such as File together with LocalStorage, LocalDatabase, Configuration or Settings store their data on an internal storage partition of the HUB-GM100. The path depends on the Application.name property, i.e. all files are stored in the `/storage/incore/<APPLICATION NAME>/` directory. If Application.name is not set, `DefaultApp` is used, so you'll find all files in the `/storage/incore/DefaultApp/` directory.

To examine the files on the console, log in via PuTTY and navigate to the corresponding directory via `cd`. Now you can inspect the files using appropriate Linux tools such as `view`, `head/tail` or `grep`. See 10 command-line tools for data analysis in Linux for more information.

The files in question can also be transferred to your local computer using the Secure Copy Protocol (SCP). For this purpose, the SSH service must first be started, e.g. by **putting the device into development mode**. On Linux you can then use `scp` to copy the files:

```
scp incore@192.168.123.1:/storage/incore/<APPLICATION NAME>/* .
```

On Windows download, install and run WinSCP. In the **Login** dialog

- change the **File protocol** to **SCP**
- enter `192.168.123.1` (or the corresponding IP address of the device in your LAN) into the **Host name** field
- set both the username to `incore` and the password to `incore`

Press **Save** to remember the settings for the next session. Now log in using the **Login** button (Fig. 5.1).

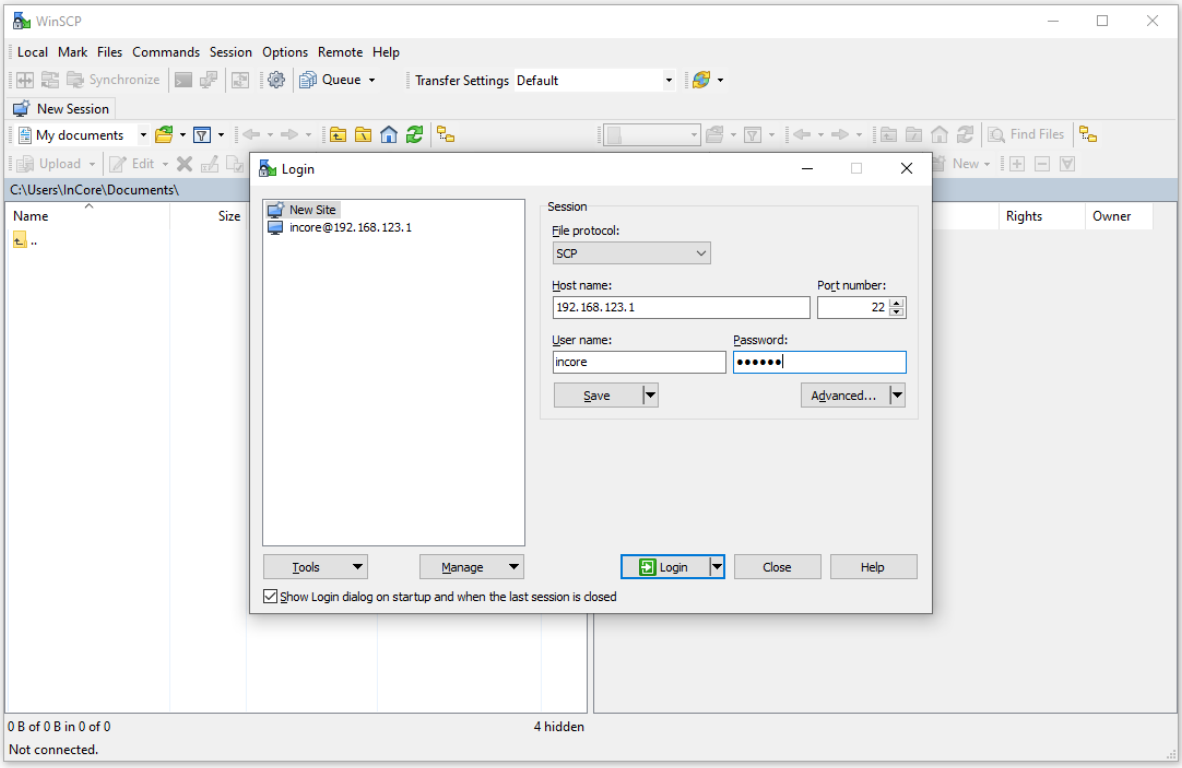

Figure5.1: Logging in with WinSCP

In the appearing dialog window confirm the server's host key initially by pressing **Yes**. Now you can navigate to the directory `/storage/incore` and access your files. Files can be copied by pressing `F5` (Fig. 5.2) and removed by pressing `Del`.

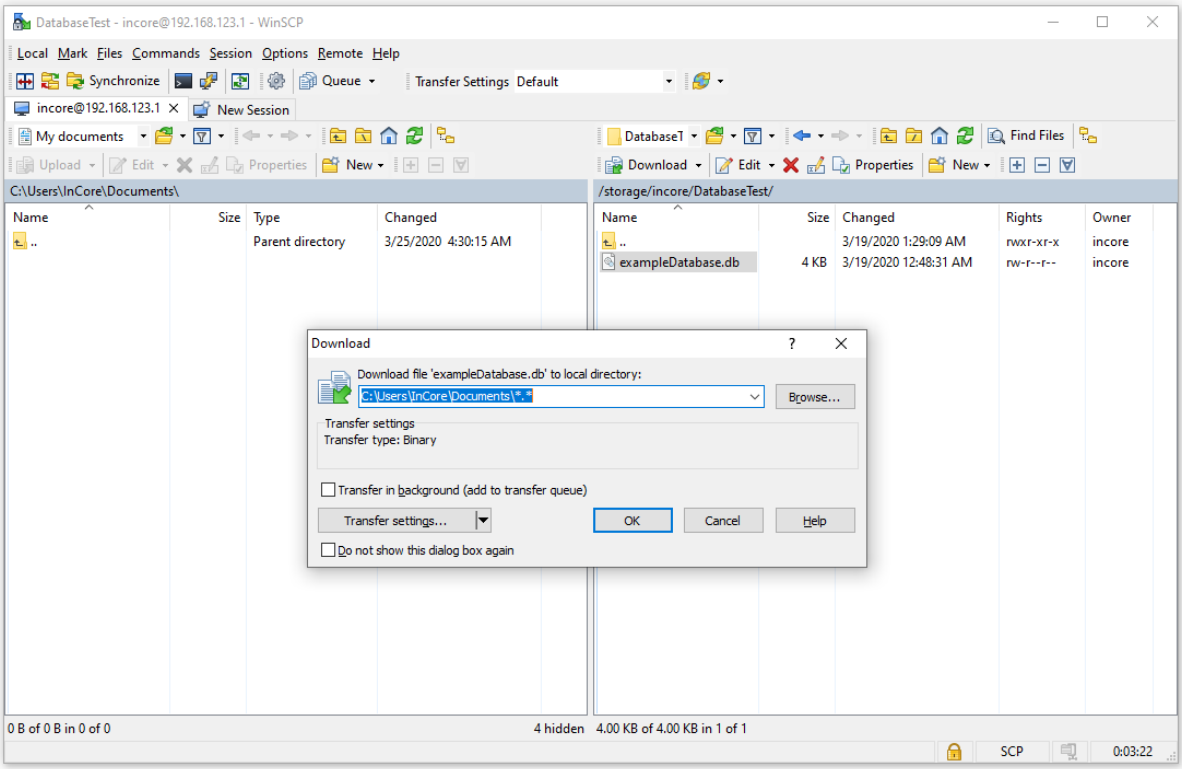

Figure5.2: Copy files to local computer via WinSCP

6 Glossary

IDE Integrated Development Environment

IoT Internet of Things

IIoT Industrial Internet of Things

LED Light-Emitting Diode

PC Personal Computer

QML Qt Modeling Language

SDK Software Development Kit

SSH Secure Shell